Notes Collection of Nginx

Nginx

THEQIONG.COM

穷屌丝联盟

Nginx Notes

穷屌丝联盟

2017 年 2 月 15 日

目录

插图

表格

Part I

Intro

Chapter 1

Overview

1.1 Nginx

Nginx 是一个用于反向代理 HTTP, HTTPS, SMTP, POP3, IMAP 的协议链接的 Web 服务器软件，同时还是一个负载均衡器和一个 HTTP 缓存服务器。

作为一个面向性能设计的 HTTP 服务器，Nginx 相比 Apache、lighttpd 具有占有内存少，稳定性高等优势。例如，与旧版本（<=2.2）的 Apache 不同，nginx 不采用每客户机一线程的设计模型，而是充分使用异步逻辑，削减了上下文调度开销，所以并发服务能力更强。

Nginx 整体采用模块化设计，其 http 服务器核心功能也是一个模块。

Nginx 有丰富的模块库和第三方模块库，配置灵活。以前，旧版本的 nginx 的模块是静态的，添加和删除模块都要对 nginx 进行重新编译，1.9.11 以及更新的版本已经支持动态模块加载。

在 Linux 操作系统下，nginx 使用 epoll 事件模型，得益于此，nginx 在 Linux 操作系统下效率相当高。例如，Nginx 在官方测试的结果中能够支持五万个并行连接，而在实际的运作中，可以支持二万至四万个并行链接。

Nginx 在 OpenBSD 或 FreeBSD 操作系统上采用类似于 epoll 的高效事件模型 kqueue。

Nginx 和 PHP-FPM 的组合是一种稳定、高效的 PHP 运行方式，效率要比传统的 Apache 和 mod_php 高出不少。其中，PHP-FPM 从 PHP5.3.3 开始合并到 PHP 核心，编译时加上`--enable-fpm`即可提供支持。

在 PHP-FPM 和 Nginx 集成使用时，PHP-FPM 以守护进程在后台运行，Nginx 响应请求后，自行处理静态请求，PHP 请求则经过 fastcgi_pass 交由 PHP-FPM 处理，处理完毕后返回。

如果 PHP-FPM 崩溃退出，那么前端将失去响应，这时 Nginx 会返回“The page you are looking for is temporarily unavailable. Please try again later.”的错误信息。

1.2 Tengine

Tengine 是一个由淘宝从 nginx fork 出来的 HTTP 服务器，同时 Tengine 也不断从 Nginx 继承其更新，所以 Tengine 兼容 Nginx 最新版 1.6.2 的所有特性，与 Nginx 的配置兼容。

Tengine 从原来的 nginx 添加了下列各项内容：

- 通过对上传到 HTTP 后端服务器或 FastCGI 服务器的请求整流，以及通过增加一致性 hash 模块、会话保持模块，加上对服务器的主动健康检查，根据服务器状态而自动加添或减少服务器的实例，大量减少对服务器机器的 I/O 压力，大大增强其负载均衡能力；
- 支持动态模块加载（DSO）支持，通过把模块编译成为可共享程序库，令服务器增添模块后无需再把整个服务器程序重新编译；
- 受到 Apache HTTP Server 的 modconcat 功能启导的 CONCAT 模块，可组合多个 CSS、JavaScript 文件的访问请求变成一个请求，以减少数据流量及提高压缩比；
- 输入过滤器主体，以更方便地管理在防火墙和事件到 HTTP 级别之间的连接；
- 模块 Sysguard，限制使用的存储器或 CPU 资源时使用率超过某个阈值。

上列内容主要是从处理请求的效率及扩展性的增润，而且这些修正部分已为 nginx 主分支接纳。

```
$ sudo add-apt-repository ppa:brightbox/tengine
$ sudo apt-get update
$ sudo apt-get install tengine
```

Part II

Core

Chapter 2

Overview

```
user www www;
error_log /var/log/nginx/error.log info;
pid /var/run/nginx.pid;

worker_processes 4;
worker_priority 0;
worker_cpu_affinity 0001 0010 0100 1000;
#worker_rlimit_core
#worker_rlimit_nofile
#working_directory

daemon on;
master_process on;
debug_points stop;

env MALLOC_OPTIONS;
env PERL5LIB=/data/site/modules;
env OPENSSL_ALLOW_PROXY_CERTS=1;

#ssl_engine

#thread_pool default threads=32 max_queue=65536;
timer_resolution 100ms;

load_module modules/ngx_mail_module.so;

lock_file logs/nginx.lock;
```

```
pcre_jit on;

include /etc/nginx/modules-enable/*.conf;

events {
  use epoll;
  worker_connections 1024;
  worker_aio_requests 32;


  # multi_accept on;
  accept_mutex off;
  accept_mutex_delay 500ms;
  debug_connection 127.0.0.1;
}

http {
  # ...
}

mail {
  # ...
}

stream {
  # ...
}

google_perftools_profiles /path/to/profile;
```

Nginx 的核心指令（core directive）如下：

- accept_mutex（串行执行 worker 来接收请求）
- accept_mutex_delay（串行执行 worker 延迟）
- daemon（持久执行）
- debug_connection
- debug_points
- error_log
- env
- events

- include
- load_module
- lock_file
- master_process
- multi_accept
- pcre_jit
- pid
- ssl_engine（SSL 硬件加速器）
- thread_pool
- timer_resolution
- use
- user
- worker_aio_requests
- worker_connections
- worker_cpu_affinity
- worker_priority
- worker_processes
- worker_rlimit_core
- worker_rlimit_nofile
- working_directory

Chapter 3

accept_mutex

当一个新连接到达时，如果激活了 accept_mutex，那么多个 Worker 将以串行方式来处理，其中有一个 Worker 会被唤醒，其他的 Worker 继续保持休眠状态；如果没有激活 accept_mutex，那么所有的 Worker 都会被唤醒，不过只有一个 Worker 能获取新连接，其它的 Worker 会重新进入休眠状态，这就是「惊群问题」。

假设你养了一百只小鸡，现在你有一粒粮食，那么有两种喂食方法：

- 你把这粒粮食直接扔到小鸡中间，一百只小鸡一起上来抢，最终只有一只小鸡能得手，其它九十九只小鸡只能铩羽而归。这就相当于关闭了 accept_mutex。
- 你主动抓一只小鸡过来，把这粒粮食塞到它嘴里，其它九十九只小鸡对此浑然不知，该睡觉睡觉。这就相当于激活了 accept_mutex。

Nginx 缺省激活了 accept_mutex，虽然说不会有惊群问题，但是这是一种保守的选择。

Apache 动辄就会启动成百上千的进程，如果发生惊群问题的话，影响相对较大；但是对 Nginx 而言，一般来说，worker_processes 会设置成 CPU 个数，所以最多也就几十个，即便发生惊群问题的话，影响相对也较小。

即使关闭了 accept_mutex，仍然可能会引起一定程度的惊群问题，表现为上下文切换增多（sar -w）或者负载上升，如果网站访问量比较大，为了系统的吞吐量，还是建议大家关闭 accept_mutex。

修改一下问题的场景，不再只有一粒粮食，而是一盆粮食时，如果仍然采用主动抓小鸡过来塞粮食的做法就太低效了，一盆粮食不知何年何月才能喂完，可以设想一下几十只小鸡排队等着喂食时那种翘首以盼的情景，此时更好的方法是把这盆粮食直接撒到小鸡中间，让它们自己去抢，虽然这可能会造成一定程度的混乱，但是整体的效率无疑大大增强了。

和激活 accept_mutex 相比，关闭 accept_mutex 的时候，请求在多个 worker 间的分配

更均衡。

- 开启 accept_mutex 时：

```
shell> ./ngx-req-distr -m `cat /path/to/nginx.pid`
Tracing 12970 12971 12972 12974 (/path/to/nginx)...
Hit Ctrl-C to end.
^C
worker 12970: 0 reqs
worker 12971: 37 reqs
worker 12972: 127 reqs
worker 12974: 3 reqs
```

- 关闭 accept_mutex 时：

```
shell> ./ngx-req-distr -m `cat /path/to/nginx.pid`
Tracing 20433 20434 20435 20436 (/path/to/nginx)...
Hit Ctrl-C to end.
^C
worker 20433: 75 reqs
worker 20434: 32 reqs
worker 20435: 29 reqs
worker 20436: 44 reqs
```

Chapter 4

accept_mutex_delay

在设置了 `accept_mutex on;` 后，最好设置下 `accept_mutex_delay 50ms;`。

Chapter 5

daemon

Chapter 6

debug_connection

Chapter 7

debug_points

Chapter 8

error_log

Chapter 9

env

Chapter 10

events

Chapter 11

include

Chapter 12

load_module

Chapter 13

lock_file

Chapter 14

master_process

Chapter 15

multi_accept

multi_accept 告诉 Nginx 收到一个新连接通知后接受尽可能多的连接。

Chapter 16

pcre_jit

Chapter 17

pid

Chapter 18

ssl_engine

Chapter 19

thread_pool

Chapter 20

timer_resolution

Chapter 21

use

Chapter 22

user

Chapter 23

worker_aio_requests

Chapter 24

worker_connections

Chapter 25

worker_cpu_affinity

Chapter 26

worker_priority

Chapter 27

worker_processes

Chapter 28

worker_rlimit_core

Chapter 29

worker_rlimit_nofile

Chapter 30

working_directory

Chapter 31

http

Chapter 32

mail

Chapter 33

stream

Part III

HTTP

Chapter 34

Overview

Nginx 可以反向代理 HTTP 协议请求。

Part IV

HTTPS

Chapter 35

Overview

Nginx 可以反向代理 HTTPS 协议请求。

Part V

SMTP

Chapter 36

Overview

Nginx 可以反向代理 SMTP 协议请求。

Part VI

POP3

Chapter 37

Overview

Nginx 可以反向代理 POP3 协议请求。

Part VII

IMAP

Chapter 38

Overview

Nginx 可以反向代理 IMAP 协议请求。

Part VIII

Load Balance

Chapter 39

Overview

Nginx 可以实现负载均衡。

Part IX

CDN

Chapter 40

Overview

Nginx 可以实现 HTTP 缓存和 CDN 功能。

Part X

URL Rewrite